Journal of Korean Institute of Intelligent Systems
Vol. 25, No. 6, December 2015, pp. 542-550
http://dx.doi.org/10.5391/JKIIS.2015.25.6.542

ISSN(Print) 1976-9172
ISSN(Online) 2288-2324

JKIIS

불량 조기 검출을 위한 SSD 테스트 케이스 개발 방법

SSD Test case generation method for early defect detection

손명규* · 이지형*†
Myeong-Gyu Son and Jee-Hyong Lee†

*성균관대학교 정보통신대학
College of Information and Communication, Sungkyunkwan University

요 약

일반적으로 새로운 SSD(Solide State Drive)를 개발할 때는 이전 세대 제품의 소프트웨어 플랫폼을 재사용하게 된다. 따라서, 이전 세대 제품을 평가할 때 사용했던 동일 테스트 케이스를 이용하여 새로운 제품을 평가하면 여러 가지 이점이 있을 수 있지만, 우선순위 또는 가중치가 고려되지 않음으로 인해 평가 리소스의 사용에 비효율성이 존재하게 된다. 이와 같은 시간적, 공간적인 리소스 낭비에 의한 비효율 발생을 방지하기 위해 새로운 방법을 제안한다. 이전 세대 제품의 평가 데이터 분석을 통해서, 불량을 검출해 낼 수 있는 가장 높은 확률의 테스트 케이스의 조합을 찾아내고, 이를 활용함으로써, 평가에 사용되는 리소스의 낭비를 최소화 시키는 방법이다. 소프트웨어가 재사용될 경우, 플랫폼 코드를 베이스로 두고, 수정 또는 추가되는 모듈의 코드가 플랫폼 코드에 통합되는 특징을 가진다. 이러한 특징 때문에 코드가 통합되는 부분에 이전과 유사한 타입의 불량이 다수 존재하게 되는 것이며, 기존 평가에서 가장 높은 확률의 불량 검출율을 가진 평가 조합을 검증에 적용함으로써 내제되어 있는 불량을 조기에 검출할 수 있게 되는 것이다. 이와 같이, 다음 세대 제품의 불량들을 조기에 검출할 수 있다면, 이는 불량 개선에 드는 비용을 최소화 시킬 수 있다는 것을 의미한다.

키워드 : SSD, 테스트 케이스, 불량 검출, 블랙박스 테스팅, 경험 기반 테스팅

Abstract

Usually, a new SSD (Solide State Drive) product is developed based on the software platform of the previous product. Therefore, when using the same test case was used to evaluate the previous generation evaluation of new products are a number of advantages may be, but a priority or weight is the inefficiency exists in the use of the evaluation resources due to not considered. A new method is proposed to prevent the waste of testing resources. Through the analysis of the evaluation data for the previous products, the combinations of testing cases with the highest probability for defect detections are identified. When the software is to be reused, most part of the base software platform is rarely modified and only some modules are added or modified. So, the whole software system may have similar types of defects with the previous products. By utilizing the evaluation data for the previous proucts, we can detect defects at an early stage.

Key Words : SSD, Test Case, Defect detection, Black box testing, Experience based testing

Received: Sep. 16, 2015
Revised : Oct. 23, 2015
Accepted: Oct. 25, 2015
†Corresponding author
John@skku.edu

1. 서 론

전통적인 방식의 HDD가 플레터(Plater)와 헤드(Head)의 조합으로 물리적인 이동을 통해 원하는 주소를 찾아가는데 반해, SSD는 논리적인 주소와 물리적인 주소간의 맵핑을 통해 전기적으로 원하는 주소를 찾아간다는 장점을 가지고 있다. 이와 같은 특성으로 인해 SSD는 HDD 대비 상대적으로 빠르고 가볍다는 제품적인 특징을 가지게 되고, 이 때문에 시장에서 사용자들은 더 많은 비용을 지불하더라도 SSD를 선호하는 경향을 보이고 있다. 하지만 SSD가 이와 같은 고성능의 결과를 내기 위해서는 반드시 소프트웨어의 지원이 필요한 것이 사실이다. 때문에 SSD 제품 개발에 있어서 소프트웨어의 무결성은 제품의 품질과 직결되는 문제라고 할 수 있다. 또한 소프트웨어의 결함이 발견 되고 조치되는 시점에 따라 제품 개발 비용은 기하급수적으로 차이를 보이게 된다. 때문에 SSD 제품 개발 산업

에서의 최대 관심사는 우선 결함이 없는 제품을 만드는 것이며 결함이 있다면 최대한 조기에 결함을 검출해서 조치하는 것이 목표라고 할 수 있다 [6].

SSD와 같은 저장장치에 적용되는 소프트웨어는 이전 제품의 기능을 포함하고 있어야 하는 하위 호환성(Backward compatibility) 특징을 가진다. 그렇기 때문에 이와 같은 하위 호환성의 특징을 가지는 소프트웨어는 이전 세대의 코드를 재사용해서 쓰는 것이 일반적인 양상이다. 이와 같이, 이전 세대의 코드를 재사용하는 경우에 해당하는 제품들은 이전 세대의 특징 또한 그대로 전이 받게 된다. 이 같은 특징 전이는 불량 측면에서도 예외일 수는 없을 것이다. 다시 말하자면, 코드 재사용에 의해 개발된 제품의 경우, 이전 세대의 것과 동일한 유형의 불량을 내제하게 된다는 것이다. 그러나 또 다른 측면으로 고려해 보면, 다음 세대 제품이 내제하고 있을 것으로 예상되는 불량은 바로 이전 세대의 불량을 통해 예측해 낼 수 있다는 논리 또한 도출할 수 있게 된다. 만약, 다음 세대 제품의 불량 유형을 예측할 수 있다면, 마찬가지 논리를 통해, 이전 세대의 평가 방법에 대한 분석을 통해 각각의 불량 유형들을 검출해 낼 수 있는 특별한 평가 방법에 대해서도 예측할 수 있게 되는 것이다. 다음 세대의 제품이 가지고 있을 것으로 예상되는 불량에 대해 해당 불량들을 조기에 검출할 수 있는 환경을 구축할 수 있다면, 불량 개선을 개발 단계 초기에 진행할 수 있게 되는 것이며, 이는 불량 개선에 드는 비용을 최소화 시킬 수 있다는 것을 의미한다. 또한 불량 개선의 비용 최소화 측면에서 생각할 때, 비록 불량 검출 시간이 오래 걸린다 하더라도 개발 초기 단계에서 특정 불량을 검출해 낼 수 있는 평가 방법이 있다면 더욱 의미 있는 일이라고 할 수 있을 것이다 [7][8].

이전 세대의 소프트웨어 플랫폼을 재사용하게 되는 특성을 가지는 SSD와 같은 저장장치 제품의 품질을 검증하기 위해서는, 다음 세대 제품의 개발을 위해 소프트웨어를 재사용하는 것과 마찬가지로, 해당 제품의 평가를 위해 테스트 케이스 또한 재사용되는 것이 일반적이다. 이때 사용되는 테스트 케이스는 이전 세대 제품의 불량을 검출했던 이력이 있는 테스트 케이스 일 것이며, 이를 통해 다음 세대 제품에 내제 되어 있을 것으로 예상 되는 불량을 검출할 수 있을 것으로 예상되기 때문에 테스트 케이스 재사용을 통한 검증은 반드시 수반되어야 한다. 하지만 단순하게 동일한 테스트 케이스를 적용하여 평가를 진행한다면 몇 가지 비효율적인 결과와 직면하게 된다. 만약, 각각의 테스트 케이스에 동일한 테스트 시나리오가 포함 되어 있다면, 각각의 평가에서 동일한 불량만을 검출하게 될 것이다. 이는 불량이 다른 불량에 의해 가려져있어서 발생할 수 있는 현상이며, 첫 번째 불량을 수정하고 다시 동일한 테스트 케이스를 통해 평가한다면 두 번째 불량도 검출할 수 있을 것이다. 하지만, 이와 같이 테스트 케이스를 순환 시켜서 반복 평가 할 경우, 한번 평가 되었던 모든 테스트 케이스를 다시 재실행 시켜야 하기 때문에 두 배, 세 배의 평가 시간이 소요되게 된다. 더불어 평가를 위한 장비의 숫자가 제한되어 있다면 소요되는 시간은 더 증가하게 되는 것이다. 이와 같은 시간적, 공간적인 리소스 낭비에 의한 비효율 발생을 방지하기 위해 다음과 같은 방법을 제안한다 [9].

효율적인 평가를 위해서는 가장 효과적인 방법은 기존에 발생한 불량들을 Trigger 시키는 가장 높은 확률의 테스트 시나리오들을 찾아내어서, 찾아낸 시나리오들로 구성된 테스트 케이스를 통해 평가를 진행하는 것이다. 이와 같은 방법을 적용한 평가를 통해, 짧은 시간 안에 이전 세대 제품이 가지고 있었던 불량과 유사한 유형의 불량을 재 발생 시키거나, 또는 새로운 유형의 불량을 발생 시켜서, 평가 대상의 완성도를 향상 시킬 수 있을 것이다. 기존 불량을 Trigger 시키는 가장 높은 확률의 테스트 시나리오를 찾아내기 위해서는 먼저 불량들을 발생시키는 모든 테스트 케이스들을 테스트 개발자들이 이해할 수 있는 유닛 단위로 분류 할 수 있어야 한다 [4]. 하나의 테스트 케이스를 테스트 목적이라는 단위로 구분하기는 힘들지만 Operation이라는 단위로 세분화 할 경우 구분이 가능하게 되기 때문이다. 다음 단계로 해당 불량과 불량을 발생시킨 Operation들과의 관계를 찾아내어 그 정보를 통해 테스트 케이스의 시나리오를 설계하는데 활용하는 것이다. 특정 불량을 발생시키는 Operation들의 조합들을 찾고, 그 조합들만이 갖는 특성 분석을 진행함으로써, 불량이 발생하지 않는 조합을 찾을 수 있고, 더불어 평가되지 않았던 조합 또한 찾아 평가에 적용할 수 있게 되는 것이다 [2][5]. 이와 같은 방법을 통해, 중복되는 테스트 시나리오를 회피하고, 의미 있는 조합으로만 구성된 테스트 케이스를 개발할 수 있게 되는 것이며, 더 많은 불량을, 짧은 시간 안에, 적은 수의 장비를 통해 검출해 낼 수 있게 되는 것이다.

본 논문은 다음과 같은 순서로 구성되어 있다. 2장에서는 테스트 케이스를 개발하기 위한 일반적인 방법론과 경험기반 방법론에 대한 관련 연구에 대해 기술한다. 3장에서는 기존의 일반적, 경험적 방법론의 문제를 정의하고, 이 문제를 풀기 위한 확률 기반의 제안 기법에 대해 설명한다. 4장에서는 실제 평가 데이터를 사용한 실험 결과를 제시한다. 5장에서는 결론 및 향후 연구에 대해 기술한다.

2. 관련 연구

2.1 일반적인 기존 방법론

테스트 케이스를 개발하는 기존 방법으로는 크게 두 가지로 Block box 기법과 White box 기법이 있다. Black box 기법은 테스트 대상의 내부 구조를 참조하지 않고, 요구사항서 등과 같이 문서에 기반 하거나 또는 개발자, 테스터, 사용자들의 경험을 분석하여, 기능적 혹은 비 기능적 테스트 조건으로 테스트 케이스를 도출하는 방법이며, White box 기법은 테스트 대상이 되는 소프트웨어의 내부 구조에 바탕을 두고, 소스 코드와 소프트웨어 구현 정보를 기반으로 테스트 케이스를 도출하는 방법이다. 그림 1에 테스트 케이스 개발 방법론에 대해 정리하였다.

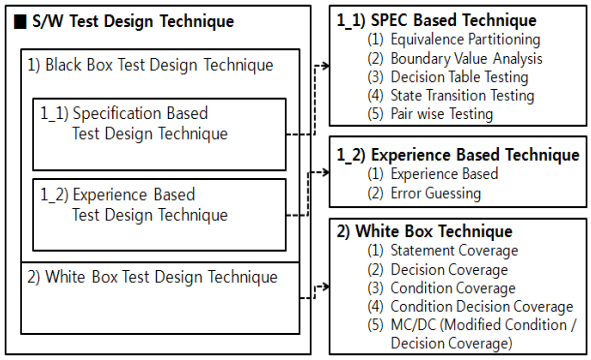

그림 1. 테스트 케이스 개발 방법론
Fig. 1. Test case development methodologies

Black box 기법을 통해 테스트 케이스 도출하는 기법 중에 가장 흔히 사용되며 효과적인 방법으로는 Pair-Wise 기법을 적용한 방법이 있다. Pair-Wise 조합을 통해 커버해야 할 기능적 범위에 비해 상대적으로 적은 량의 테스트 Set를 구성시킬 수 있는 장점이 있다. Pair-Wise 기법은 관찰 결과 대부분의 결함이 2개 이상의 요소에 상호작용에 기인한다는 것에 착안하여 2개 요소의 모든 조합을 다루는 기법이다. 다시 말해, Pair-Wise 조합의 의미는 테스트를 하는데 필요한 각 값들이 다른 파라미터 값과 최소한 한 번씩은 조합을 이룬다는 의미를 가지고 있다. 3가지의 요소에 대해 각각의 요소가 7개씩의 파라미터를 가지게 되면 7 x 7 x 7로 343가지의 테스트가 진행되어야 한다. 하지만 Pair-Wise 기법을 적용할 경우, 53가지의 테스트 케이스로 줄일 수 있다 [7].

일반적으로 Black box 기법 중 요구사항서를 기반으로 한 기능 위주의 수락 시험(Acceptance Testing)만으로는 소스 코드상의 모든 경로를 테스트하기에는 무리가 있다. 이러한 현실로 인해 소스 코드 기반의 Whit box 기법이 필요하게 되는 것이다. White box 기법에서 평가하는 대상이 되는 소스코드에 대해 얼마나 충분한 테스트를 진행하였는가를 나타내는 지표로 커버리지 라는 단위를 쓴다. 이는 말 그대로 테스트를 통해 코드가 얼마나 터치 되었는가를 뜻하며, 소스 코드상의 전체 유닛 대비 테스트를 통해 실행된 유닛의 비율을 나타낸다. 코드를 이루는 유닛은 크게 구문(Statement), 조건(Condition), 결정(Decision)이다. White box 기법의 테스트 커버리지는 유닛을 기준으로 하위 레벨의 유닛 순서로 Statement Coverage, Branch Coverage, MC/DC (Modified Condition/Decision Coverage)들이 있다. White box 기법을 통해 소프트웨어의 소스코드에 기술된 내용이 실제로 호출되는지를 확인하는 대표적인 기법으로 MC/DC 기법이 사용된다. MC/DC는 소프트웨어 분기(Branch)내에서 발생할 수 있는 모든 논리적인 조합 중 단위 조건 하나의 값으로 값이 고정된 다른 모든 것을 True 또는 False로 변경하여 전체 분기의 평가에 영향을 주는 조합을 말한다. 분기 내에서 발생할 수 있는 모든 논리적인 조합을 시험하는 조건인 Multiple Condition Coverage와 비슷한 수준의 결함 검출 능력을 가지면서 테스트 케이스의 수는 적게 산출하는 효과가 있다 [8].

위와 같은 기존의 기법들을 통해 테스트 케이스를 만들 경우, 개발 되는 테스트 케이스의 수가 무제한적으로 증가하게 된다. 100%의 커버리지를 가지는 테스트 케이스 Set의 테스트 케이스 수는 가늠하기 쉽지 않다. 소프트웨어 테스팅에 있어 충분한 테스팅에 대한 기준은 분분하다고 할 수 있다. 소프트웨어 테스팅 공학에서조차 테스트 회수는 무결점을 추구하는 것보다 개발 비용에 따른 결함 허용치를 협의해야 한다고 기술되어 있다. 다시 말하자면, 기존의 기법들의 문제점은 Cause-Effect에 의한 확률적 접근이 아니라, 명시적인 산술적 접근이라는데 있다. 또한 기존의 기법들은 명시적인 테스트 시나리오를 통해 불량을 발견 할 수는 있지만, 직관적인 시나리오를 통해 평가가 이뤄지기 때문에 해당 불량을 발생시키는데 가장 큰 영향을 미치는 Input 조합을 찾아내는 것은 힘들다.

2.2 경험 기반 방법론

최초 개발된 제품에 대한 평가 방법을 구축해야 하는 경우에는, 일반적인 Black box, White box 기법을 통해 테스트 케이스가 개발되어 평가 되지만, 유사한 기능을 가지고 있는 다음 버전 제품의 평가를 위해서는 체계적인 기법을 보강하기 위해, 일반적인 기법을 적용한 이후에 추가적으로 경험 기반 기법을 적용하는 것이 일반적인 기법으로 다루기 어려운 특별한 불량들을 찾아내는데 유용하다. 보편적으로 사용되는 경험 기반 기법으로는 탐색적 테스팅 접근법(Exploratory Testing Approach)과 오류 추정(Error Guessing) 기법이 있다.

탐색적 테스팅 접근법은 테스트의 설계, 계획, 수행, 기록 및 학습을 동시에 진행하는 발견적인 테스팅 기법이다. 테스트 케이스를 먼저 작성하지 않고, 테스트 대상 제품을 실행하면서 익숙해지는 것과 동시에 테스트를 설계하고 계획하는 작업을 진행하게 된다. 탐색적 테스팅 접근법의 순서도를 그림 2에 나타내었다. 보통 제한된 60~120분 정도의 시간 내에 테스팅의 목적을 정한 후 몰입하여 최소한의 설명 가능한 기록을 남기면서 테스트를 수행하고, 수행 후 요약 보고 하는 방식이다. 이는 일반적인 테스팅 방법보다 더 효율적이고 결함을 발견할 확률이 높아질 수 있다고 보는 접근법이다. 여러 테스팅 기법과 방법론으로 훈련되고 숙달된 엔지니어가 많은 생각과 지적인 활동을 통해 창조적으로 테스트 케이스를 개발하는 방법이라고 할 수 있다.

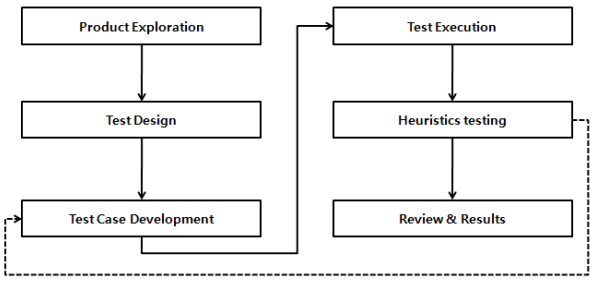

그림 2. 탐색적 테스팅 접근법
Fig. 2. Exploratory testing approach

오류 추정 기법은 테스팅을 담당하는 엔지니어가 테스트할 제품과 시스템에 대해 완전히 이해하고 있다는 전제로 적용되는 기법이며, 기본적으로는 제품의 식별된 취약점, 변경점등을 기반으로 테스트를 진행하는 방식이다. 먼저 이전 제품 A 대비, 다음 제품 B의 취약점, 변경점을 파악하여 나열한다. 그리고 난 후, 나열된 것과 같은 불량 발생 가능성이 있을 것으로 추정되는 부분에 대해 테스트 계획을 세우고 평가를 진행한다.

3. 제안 기법

Black box 기법과 White box 기법의 경우, 테스트 케이스를 개발하는 기법으로 널리 적용되어 오고 있다. 하지만 커버해야 하는 요구사항 및 소스 코드의 규모가 크고 방대한 경우, 그 규모에 비례하여 테스트 케이스의 개수도 증가하게 된다. 일반적인 산업 분야에 있어서 모든 평가를 진행하기 위해서는 그 만큼 많은 시간과 비용이 소모되게 된다. 때로는 제한된 시간과 공간을 조건으로 최적의 완성도를 가지는 평가를 해야만 하는 상황이 발생하게 되는 것이다. 그렇기 때문에 중요도를 기준으로 전문가가 판단하여 평가 항목의 우선순위를 선정하는 작업을 진행하여야 한다. 이렇게 정해진 조건하에서 평가 항목의 우선순위를 선정해야 하는 상황에서는 Black box 기법과 White box 기법으로는 효율적인 평가를 위한 테스트 케이스를 개발하기가 힘들다. 또한, 경험 기반 기법을 적용하더라도, 다양한 불량을 발생시켜서 검출해 내기 위해서는, 동일한 평가를 순환 반복하여 평가하는 방법을 사용할 수밖에 없기 때문에, 다양한 불량을 검출해내기 위해서는 평가 시간과 평가 PC 같은 리소스를 비효율적으로 사용하게 되는 결과를 초래하게 된다. 이와 같은 문제점들을 개선하기 위해 다음과 같은 기법을 제안한다. 그림 3에 제안 기법을 도식화 하였다.

효율적인 평가를 위해서는 가장 효과적인 방법은 기존에 발생한 불량들을 Trigger 시키는 가장 높은 확률의 테스트 시나리오들을 찾아내어서, 찾아낸 시나리오들로 구성된 테스트 케이스를 통해 평가를 진행하는 것이다. 이와 같은 방법을 적용한 평가를 통해, 짧은 시간 안에 동일한 불량을 재 발생시키거나, 또는 동일한 불량이 발생하지 않음을 확인하는 것으로 평가 대상의 완성도를 판단할 수 있을 것이다.

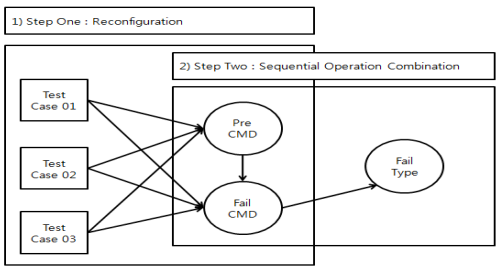

그림 3. 제안 기법 도식화
Fig. 3. The proposal scheme

기존 불량을 Trigger 시키는 가장 높은 확률의 테스트 시나리오를 찾아내기 위해서는 먼저 불량들을 발생시키는 모든 테스트 케이스들을 테스트 케이스 개발자들이 이해할 수 있는 유닛 단위로 분류 할 수 있어야 한다. 각각의 테스트 케이스들은 한 사람이 아닌 서로 다른 개발자에 의해 구현되었을 것이다. 때문에 자신이 구현한 테스트 케이스가 아닌 경우, 테스트 케이스가 어떤 목적으로 그와 같은 시나리오로 개발되었는지 파악하기 쉽지 않다. 이때, 테스트 케이스의 목적을 직관적으로 파악하기는 힘들지만 테스트 케이스가 어떤 Operation들의 순서로 구현되었는지는 파악할 수 있다. 이와 같이, 하나의 테스트 케이스를 테스트 목적이라는 단위로 구분하기는 힘들지만 Operation이라는 단위로 세분화 할 경우 구분이 가능하게 된다. 다음 단계로 해당 불량과 불량을 발생시킨 테스트 케이스의 관계를 찾아내어야 한다. 더 자세하게 말하자면, 해당 불량과 불량을 발생시킨 테스트 케이스의 Operation들과의 관계를 찾아내야 하는 것이다. A라는 불량이 있다고 가정하자. 또한 이때 A라는 불량을 Trigger하는 테스트 케이스는 Test_Case_01, Test_Case_02 2가지가 있고, Test_Case_01은 a, b, c라는 Operation에 의해 구성되어 있으며, Test_Case_02은 a, b, d라는 Operation에 의해 구성되어 있다고 가정해 보자. 이와 같은 경우 A라는 불량을 Trigger하는 가장 높은 확률의 Operation 조합은 a와 b를 조합한 경우라는 정보를 얻어 낼 수가 있다. 동일한 방법으로 모든 불량에 대해 해당 불량을 Trigger 시키는 가장 확률이 높은 Operation들을 조합을 찾아 낼 수가 있을 것이며, 이 조합의 조건들이 테스트 케이스의 시나리오를 설계하는데 활용되게 되는 것이다.

3.1 데이터 재구성

제품을 평가하기 위한 테스트 케이스들은 서로 다른 목적을 가지고 서로 다른 개발자들에 의해 구현 된다. 서로 다른 목적으로 개발되었기 때문에 테스트에 사용되는 시나리오는 동일한 패턴을 가지지 않는다. 자신이 개발한 테스트 케이스의 경우 어떤 근거를 통해 시나리오를 설계하여 테스트 케이스를 개발하였는지 명백하게 명시할 수 있으나, 타인이 개발한 테스트 케이스의 경우 소스 코드를 확인하더라도 어떤 근거를 가지고 테스트 케이스 시나리오가 설계 되었는지 파악하기는 쉽지 않다. 이렇게 차이점을 파악하기 어려운 테스트 케이스라는 Input을 분류의 기준으로 선정하게 되면 Input들 간의 중복되는 요소로 인해 분류가 명확하게 되지 않을 수 있다. 때문에 차이점을 파악할 수 있는 Input으로 초기 Input을 재구성(Reconfiguration) 할 필요가 있다.

SSD의 동작 원리는 다음과 같다. Host로부터 SSD를 동작시키기 위해 Request를 보내면, Device, 즉 SSD는 이 Request를 받아서 정해진 동작을 수행하게 된다. 이와 같은 SSD 평가를 위한 테스트 케이스는 Request에 해당하는 Command Operation과 Event Operation이 특정한 순서로 조합되어 있는 구조로 구현되게 된다. 이와 같은 구조로 구현되어 있는 테스트 케이스는 Command와 Event 각각의

Operation으로 재구성 할 수 있다. 또한 각각의 Command와 Event에 대해서는 파라미터 값을 변경해 줄 수가 있다. 파라미터 값을 변경해 줄 경우 같은 Command 일지라도 SSD에 요청하는 Request의 성격이 달라지므로 SSD 입장에서는 전혀 다른 동작을 실행하는 결과를 가져오게 된다. Event의 경우 또한 마찬가지로 파라미터 값이 변경됨에 따라 SSD에 전혀 다른 예외상황을 발생시키게 된다. 결과적으로 SSD 테스트 케이스는 파라미터로 구분된 Command와 파라미터로 구분된 Event의 순차적인 조합으로 재구성 할 수 있는 것이다.

3.2 불량별 순차적 원인 조합

이와 같이 테스트 케이스 재구성 과정을 거친 후 다음 단계로 특정 불량과 불량을 발생시킨 원인과의 관계를 찾아내는 과정을 거치게 된다. 불량을 발생시키는 원인들의 기본 단위는 테스트 케이스 재구성 과정에서 얻은 파리미터로 구분된 Command와 파라미터로 구분된 Event들이다. 하나의 테스트 케이스는 여러 가지의 Command와 Event들의 순차적인 조합으로 구성되어 있다. 이와 같은 테스트 케이스를 통해 평가를 진행 중에 어떤 불량이 발생하였다면, 불량이 발생한 시점과 일치 되는 Command 또는 Event가 있을 것이다. 이와 같이 시점이 일치되는 Command 또는 Event를 Trigger Operation 이라고 하자. 또한 불량 발생한 시점과 일치되는 Operation의 바로 이전에 실행된 Command 또는 Event가 있을 것이다. 이와 같이 바로 이전에 실행된 Command 또는 Event를 Previous Operation 이라고 하자. Previous Operation 은 다음 Operation인 Trigger Operation에 영향을 미치게 되고, 이렇게 영향을 받은 Trigger Operation은 불량을 발생시키게 된다. 이 때 모든 Trigger Operation과 Previous Operation이 특정 불량과만 일치되지는 않을 것이다. 그렇기 때문에 특정 불량과 100% 일치되는 것이 아닌 가장 높은 확률로 일치 되는 Trigger Operation과 Previous Operation의 조합을 찾아야 한다.

불량 Set D={A, B, C, D, E}가 있고, Operation Set O={a, b, c, d, e}가 있다고 가정하자. 또한 테스트를 통해 얻은 Operation Set과 불량의 관계에 대한 결과 레포트가 다음과 같이 8개가 있다고 가정하자. R={a→A, b→B, c→A, d→C, e→A, b→B, e→A }. 8개의 결과를 불량 별로 정리하면, A={a→A, c→A, e→A, e→A}, B={b→B, b→B}, C={d→C}로 정리할 수 있다. 위의 정보를 데이터만 기준으로 분석해 보면 먼저 불량 B를 발생시키는 Operation Set은 b이고, 불량 C를 발생시키는 Operation Set은 d이다. 하지만 불량 A를 발생시키는 Operation Set은 a, c, e가 있다. 이중에 가장 높은 확률로 불량 A를 발생시키는 Operation Set은 50%의 확률을 가지는 e라는 정보를 얻을 수 있다. 추가적으로 a와 c는 25%의 확률로 불량 A를 발생시킨다는 정보 또한 얻을 수 있을 것이다. 이와 같은 방법으로 얻어진 Operation Set과 불량 간의 확률 정보를 통해서 특정 불량을 발생시키는 가장 높은 확률의 조건을 알아 낼 수 있으며, 반대로 특정 불량을 발생시키는데 필요 없는 조건을 확인 할 수 있게 되는 것이다.

하지만 이때 문제점은 서로 다른 불량들을 발생 시키는 가장 높은 확률의 조합이 동일할 수 있다는 점이다. 다시 말해, 특정 Operation 조합 a가 불량 A, 불량 B를 발생시키는 가장 높은 확률의 조합일 수도 있다는 말이다. 이와 같은 경우, 불량 A를 발생시키기 위한 평가에서 불량 B가 발생하는 현상이 생길 수 있게 되는 것이다. 불량을 발생시킨다는 측면에서는 유의미한 결과이겠지만, 원하는 특정 불량을 발생시키기 위해서는 불량 A와 불량 B를 발생시키는 높은 확률의 조합들 중에 중복되는 조합을 제거할 경우, 불량 A만을 잘 발생 시키는 조합과 불량 B만을 잘 발생시키는 조합을 구할 수 있게 된다. 하지만 만약 제거된 조합이 불량을 발생키는 높은 확률의 조합일 경우, 해당 특정 불량을 발생시키는 확률이 감소하게 되고, 이로 인해 테스트 되는 시간이 길어지게 된다는 단점이 발생하게 된다.

Operation 조합은 Previous Operation과 Trigger Operation으로 구성된다. 여기서 불량을 직접적으로 일으키는 Operation은 Trigger Operation이지만, 불량 상태를 발생시키기 위한 기반 조건을 만들어 주는 것은 Previous Operation이다. 다시 말하면, Previous Operation이 무엇이냐에 따라 특정 불량이 발생할 것인지 아닌지가 결정될 확률이 높다는 것이다. 그렇기 때문에 Previous Operation의 파리미터를 변경해 주면서 평가의 다양성을 부여한다면, 불량을 발생시키는 더 효율적인 파라미터를 찾을 수 있게 되는 것이다.

4. 실험 결과

실험을 위해 SSD를 동작 시키는 소프트웨어를 대상으로, 6개월간 평가된 평가 레포트를 수집하였다. 평가 레포트 내용 가운데, 평가에 사용된 테스트 케이스와 평가 결과로 검출된 불량의 조합을, 실험을 위한 데이터로 활용하였다. 총 48가지의 테스트 케이스로 평가가 진행 되었으며, 검출된 불량의 유형을 분류하면 4가지 타입의 불량으로 분류할 수 있다. 48 가지의 테스트 케이스는 제안 기법에 명시된 것과 같이 데이터 재구성 작업을 진행하였고, 재구성을 통해, 46 가지의 Previous Operation과 20 가지의 Trigger Operation으로 재구성할 수 있다.

재구성된 Previous Operation과 Trigger Operation의 조합들 중에 분류된 각각의 불량 타입을 발생 시키는 모든 조합을 기반으로, 특정 불량을 발생시켜서 검출해 낼 수 있을 것으로 예측되는 테스트 케이스를 불량별로 개발하였다. 다시 말하자면, Defect A를 발생시켰던 모든 조합을 찾아 낸 후, 그 모든 조합들로 구성된 Test Case A를 만드는 것이다.

불량 유형을 분류하고 테스트 케이스를 작성하기 위해 이전 세대의 평가 레포트가 활용되었고, 실제 실험은 다음 세대의 제품에 대해 평가가 진행되었다.

본 논문에서 제안한 것과 같이, 데이터 재구성 작업과 높은 빈도의 Operation 조합을 선정하는 작업을 적용한 경우와 제안 방법을 적용하지 않은 경우의 불량 검출율 비교를 위해

다음과 같은 실험을 진행하였으며, 그 결과를 그림 4에 정리하였다. 테스트 케이스를 가공하지 않고 이전 세대에서 사용되었던 테스트 시나리오 그대로 사용하여 평가한 결과를 Previous Method로 표시하였으며, 테스트 케이스를 제안된 방법인 데이터 재구성을 통해 Operation단위로 변환 시키고, 그 Operation 단위의 조합들 중에 특정 불량을 발생시키는 확률이 높은 조합들로 분류하고, 분류된 조합들을 포함하고 있는 새로운 테스트 케이스를 개발하여 평가한 결과를 New Method로 표시하였다. X축은 평가에 사용된 총 시간을 의미하며, Y축은 평가를 통해 검출된 불량의 타입별 개수를 의미한다. 그리고 Detection Ratio는 Previous Method 대비 New Method의 불량 타입별 검출율 향상 수준을 나타내었다. New Method의 불량 타입별 검출율 향상 수준을 백분율로 표기하면 그림 5와 같이 나타낼 수 있다.

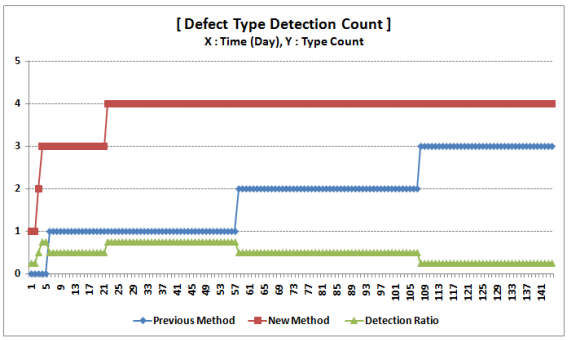

그림 4. 불량 타입별 검출 횟수
Fig. 4. Defect type detection count

Previous Method가 사용된 평가의 진행 사항은 다음과 같다. 48개의 테스트 케이스에 대해 각각 1시간씩 평가를 진행하였고, 이때 검출된 불량을 수정 후, 다시 동일한 48 개의 테스트 케이스를 반복하여 평가를 진행하였다.

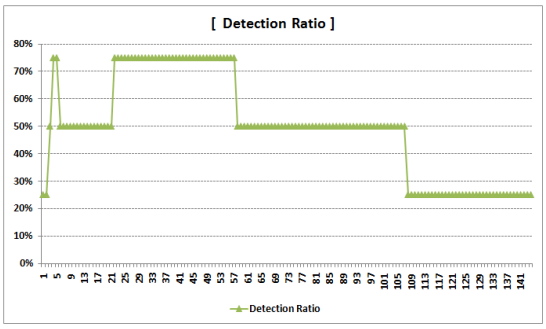

그림 5. 불량 검출 비율
Fig. 5. Defect detection ratio

이와 같은 평가를 반복적으로 3회 실시하였으며 총 평가 시간은 48 x 3으로 144시간이 소요되었으며, 검출된 불량 타입별 개수는 3가지이다. New Method가 사용된 평가는, 제안된 방법을 통해 만들어진 4개의 새로운 테스트 케이스에 대해 각각 1시간씩 평가를 진행하였고, 서로 다른 불량 타입이 검출되는 현상을 확인한 후, 동일한 평가를 15회 반복하였으며 4 x 15으로 60 시간이 소요되었으며, 검출된 불량 타입별 개수는 4가지이다.

Previous Method는 144시간 동안 3가지 타입의 불량을 검출하였고, New Method는 22시간 동안 4 가지 타입의 불량을 모두 검출할 수 있다는 결과를 확인 할 수 있다. 위의 결과를 통해 제안된 기법을 사용할 경우, 제품 평가 시간을 85% 단축시키는 효과를 가져 올 수 있게 되는 것이다.

Previous Method가 위의 경우와 같이 불량을 검출하는데 많은 시간이 소요되는 이유는 각각의 테스트 케이스들이 최적화 되어 있지 않고, 중복되는 시나리오를 가지고 있기 때문이라고 볼 수 있다. 또한 각각의 타입별 불량들을 Trigger 시키기 위한 Operation 조합들에 대한 우선순위 또는 가중치가 고려되어 있지 않기 때문이다.

표 1의 결과는, 그림 4의 New Method에 대한 세부 평가 결과이다. 4가지 타입의 불량에 대해, 4가지의 테스트 케이스가 각각 개발 되었으며, 이렇게 개발된 TC(Test Case) A, TC B, TC C, TC D를 이용하여 각각 15회 동안 평가가 진행 되었다. 세로축은 테스트 케이스의 타입을 나타내고, 가로 축은 불량의 타입을 나타낸다. 그리고 Not Failed는 하루 동안 평가 되었지만 어떤 불량도 발생하지 않은 경우를 의미한다. Match Rate은 Not Failed를 포함한 경우에 테스트케이스가 목표한 불량을 발생시킨 확률을 의미하며, Match Rate(w/o Not Failed)는 Not Failed를 제외한 경우에 테스트 케이스가 목표한 불량을 발생시킨 확률을 의미한다. 평가 결과 확인 시, 각각의 테스트 케이스가 목표한 불량만을 100% 발생 시키지는 않지만, 일부 불량의 경우 테스트 케이스가 50% 이상의 확률로 목표한 불량을 발생시키고 있음을 알 수 있다. 또한 표 1을 통해 알 수 있는 중요한 정보는 TC A와 TC B가 Defect A를 집중적으로 발생시킨다는 것과, 마찬가지로, TC C와 TC D가 Defect C를 비교적 많이 발생시키는 현상을 확인 할 수 있다는 것이다. 이와 같은 결과가 의미하는 바는, 기본적으로 Defect A와 Defect B, 그리고 Defect C와 Defect D가 유사한 성격을 가지는 불량이며 이를 발생시키는 평가 조합이 유사하다는 것을 의미한다. 다시 말해, Defect C는 TC C에 의해서도 TC D에 의해서도 빈번히 발생될 수 있으며, TC C와 TC D는, TC A와 TC B와는 평가 조합 측면에서 차별화 되지만, 서로 간은 동일한 조합을 포함하고 있다는 것을 의미한다.

표 1의 결과에서 알 수 있듯이, TC A를 통해 불량 A와 불량 B를 모두 검출 할 수 있는 효과를 얻을 수 있음이 확인되었다. 이는 TC A, B, C, D의 선별된 조합이 각각의 목표한 불량을 발생시키는 데는 높은 기여도를 가지지만, 전체적으로 볼 때는 각각의 조합이 다른 불량을 발생시키는데도 높은 기여도를 가질 수 있기 때문이다. 실험을 통해 실제로 불량 발생에 높은 기여도를 가졌지만, 목표한 불량이 아닌 다른 불량을 발생 시키는 조합과 중복되는 조합을 찾아낸 후, 테스트 케이스의 평가 시나리오에서 제외 시켜 평가해 보았을 때, 다

표 1. 테스트 케이스별 불량 검출 결과
Table 1. Defect detection results of each test case

		Not Failed	Defect A	Defect B	Defect C	Defect D	Total	Match Rate	Match Rate (w/o Not Failed)	Elapsed Days
TC A	All (Pre)	7	5	2	1	0	15	33%	63%	1.000
TC B	All (Pre)	7	5	3	0	0	15	20%	38%	1.000
TC C	All (Pre)	3	2	0	8	2	15	53%	67%	1.000
TC D	All (Pre)	2	0	1	6	6	15	40%	46%	1.000

표 2. 필터링 후 테스트 케이스별 불량 검출 결과
Table 2. Defect detection results of each test case after filtering

		Not Failed	Defect A	Defect B	Defect C	Defect D	Total	Match Rate	Match Rate (w/o Not Failed)	Elapsed Days
TC A	Filtered	4	6	0	0	0	10	60%	100%	6.200
TC B	Filtered	9	0	1	0	0	10	10%	100%	6.000
TC C	Filtered	2	0	0	8	0	10	80%	100%	5.375
TC D	Filtered	4	0	0	0	6	10	60%	100%	6.167

표 3. 파라미터 확장 적용 후 테스트 케이스 A, B의 불량 검출 결과
Table 3. Defect detection results of test cases A and B after parameter expansion

		Not Failed	Defect A	Defect B	Defect C	Defect D	Total	Match Rate	Match Rate (w/o Not Failed)	Elapsed Days
TC A	All (Pre)	7	5	2	1	0	15	33%	63%	1.000
TC B	All (Pre)	7	5	3	0	0	15	20%	38%	1.000
TC A	Filtered	4	6	0	0	0	10	60%	100%	6.200
TC B	Filtered	9	0	1	0	0	10	10%	100%	6.000
TC A	PreParam Only	3	3	2	1	1	10	30%	43%	4.857
TC B	PreParam Only	4	2	1	1	2	10	10%	17%	5.200
TC A	TrigParam Only	5	5	0	0	0	10	50%	100%	5.800
TC B	TrigParam Only	9	0	1	0	0	10	10%	100%	6.000

표 4. 파라미터 확장 적용 후 테스트 케이스 C, D의 불량 검출 결과
Table 4. Defect detection results of test case C and D after parameter expansion

		Not Failed	Defect A	Defect B	Defect C	Defect D	Total	Match Rate	Match Rate (w/o Not Failed)	Elapsed Days
TC C	All (Pre)	3	2	0	8	2	15	53%	67%	1.000
TC D	All (Pre)	2	0	1	6	6	15	40%	46%	1.000
TC C	Filtered	2	0	0	8	0	10	80%	100%	5.375
TC D	Filtered	4	0	0	0	6	10	60%	100%	6.167
TC C	PreParam Only	1	1	1	5	2	10	50%	56%	3.667
TC D	PreParam Only	1	2	1	2	4	10	40%	44%	4.111
TC C	TrigParam Only	2	0	0	8	0	10	80%	100%	5.250
TC D	TrigParam Only	3	0	0	0	7	10	70%	100%	5.714

른 불량이 아닌 목표한 불량만을 발생시키는 결과를 확인 할 수 있었다. 표 2는 TC A와 TC B, TC C와 TC D에서 높은 기여도를 가진 조합이지만 서로간의 시나리오에 중복되는 조합을 제거한 후 평가한 결과이다. Match Rate (w/o Not Failed) 값을 보면, 중복되는 조합을 제거할 경우 목표한 불량을 발생시킬 수 있다는 결과를 알 수 있지만, Elapsed Days 값과 같이 해당 불량을 검출하는데 많은 시간이 소요됨을 알 수 있다. 이때 평가 기간은 최대 7일까지 진행하였다.

이때, 필터링 이후의 개별 테스트 케이스들의 Operation 조합들을 비교해 보면, Previous Operation과 Trigger

Operation이 서로 크게 중복되지 않고 다르게 구성되어 있음을 확인 할 수 있었다. 또한 각각의 Operation 들이 가질 수 있는 파라미터는 여러 가지가 있는데도 불구하고, Operation 조합을 보면 소수의 파라미터들만이 선택 되어 있음을 확인 할 수 있다. 이와 같은 현상으로 미루어 볼 때, 이전 세대의 테스트 케이스들에 의해 터치 되지 않은 파라미터를 가지는 Operation들이 존재한다는 것을 추론할 수 있다. 다시 말해, 평가를 통해 검증되지 않은 Hidden 조합이 존재할 수 있다는 가정을 세울 수 있으며, 표 3과 표 4의 실험을 통해 실제 Hidden 조합이 존재함을 평가해 보았다.

표 3과 표 4는 각각 Previous Operation과 Trigger Operation의 파라미터들을 확장 시켜서 평가한 결과이다. 세로축의 PreParam Only가 의미하는 것은 Previous Operation의 파라미터만을 확장해서 평가한 결과임을 의미하며, TrigParam Only가 의미하는 것은 Trigger Operation의 파라미터만을 확장해서 평가한 결과임을 의미한다. 해당 실험에서 얻을 수 있는 결과는 Trigger Operation의 파라미터 확장을 통해서는 크게 다른 결과를 얻을 수 없지만, Previous Operation의 경우는 불량 검출율이 달라짐을 확인 할 수 있다.

Previous Operation의 파라미터 확장을 포함하여 평가를 진행할 경우, 목표한 불량을 검출하는 Match Rate은 떨어지지만, 불량을 검출하는 Elapsed Days는 단축되고, 또한 모든 타입의 불량이 검출 되는 결과를 확인 할 수 있다. 이와 같은 결과가 의미하는 것은, 실험 전 추론한 것과 같이 파라미터 확장을 통해서, 다른 불량을 추가적으로 검출할 수 있는 Hidden 조합을 찾아 낼 수 있다는 것을 의미한다. 모든 Operation의 모든 파라미터 조합을 통해 평가를 할 수 없는 제한된 상황에서는 이와 같은 접근법을 통해, 여러 가지 불량을 짧은 시간 안에 검출할 수 있는 평가 방법을 구축 할 수 있을 것이다.

지금까지의 실험 결과를 정리하면 다음과 같다. 먼저, 이전에 사용되었던 테스트 케이스에서 불량 검출 확률이 높은 조합을 찾아내서, 이와 같이 우리가 알고 있는 높은 확률의 조합을 통해 대표적인 불량 타입을 검출한다. 그 다음으로, 작성된 테스트 케이스의 시나리오에서 중복되지 않는 조합들을 찾아내고, 그 조합들의 파라미터 조건을 확장하여 평가를 진행함으로써, 평가되지 않았던 Hidden 조합을 찾아 낼 수가 있게 되고, 더 다양한 불량을 검출할 수 있게 되는 것이다.

5. 결론 및 향후 연구

본 논문에서는 소프트웨어를 사용하는 제품의 불량을 더 효율적으로 검출해 내기 위해, 이전 세대의 평가에 사용된 테스트 케이스를 기반으로 데이터를 재구성 시키고, 재구성된 데이터를 이용하여 불량 발생 확률이 높은 조합을 찾아내서, 그 조합으로 이루어진 테스트 케이스를 평가에 활용하는 방식을 제안하였다. 실험 결과 불량 발생 확률이 높은 조합들로만 구성된 테스트 케이스가, 일반적인 테스트 케이스에 비해, 더 다양한 불량을, 더 짧은 시간 안에 검출해 낼 수 있다는 결론을 확인 할 수 있었다. 또한, 불량을 발생시키는 Hidden 조합이 존재하며, 이는 일반적인 테스트 케이스에는 적용되지 않은 것으로, Hidden 조합을 찾아 평가에 활용할 경우, 더 다양한 불량을 검출해 낼 수 있다는 결과를 얻을 수 있었다. 이와 같은 소프트웨어 불량을 조기 검출하기 위한 평가 조건을 찾을 수 있는 방법은 하위 호환성을 가지는 제품의 품질 검증 분야에서 널리 활용될 수 있을 것이다.

향후에는 Previous Operation과 Trigger Operation의 이중 조합뿐 만이 아니라, 확장된 범위의 다중 조합에 대한 불량 발생률을 연구하여, 평가 방법과 불량 사이의 보다 더 정밀한 상관관계를 분석할 예정이다.

References

[1] C. Bishop, "*Pattern Recognition and Machine Learning*", Springer 2006.

[2] Karel Dejaeger, Thomas Verbraken and Bart Baesens, "*Toward Comprehensible Software Fault Prediction Models Using Bayesian Network Classifiers*", IEEE Transactions of Software Engineering, 39(2), February 2013.

[3] Wang He, Wang Hao and Lin Zhiqing, "*Improving Classification Efficiency of Orthogonal Defect Classification Via a Bayesian Network Approach*", 2009 Computational Intelligence and Software Engineering, CiSE 2009, 2009.

[4] Hao Tang and Shi Liu, "*Basic Theory of Fuzzy Bayesian Networks and Its Application in Machinery Fault Diagnosis*", Fourth International Conference on Fuzzy Systems and Knowledge Discovery, *FSKD 2007*, 2007, 4:132-137.

[5] DAhmet Okutan and Olcay Taner Yildiz, "*Software defect prediction using Bayesian networks*", *Empirical Software Engineering*, 19(1), February 2014.

[6] Unchalisa Taetragool and Tiranee Achalakul, "*Method for failure pattern analysis in disk drive manufacturing*", International *Journal of Computer Integrated Manufacturing,* 24(9), September 2011.

[7] Yingxia Cui, Longshu Li and Sheng Yao, "*A New Strategy for Pair wise Test Case Generation*", 3rd International symposium on Intelligent Information Technology Application, *IITA 2009*, 2009, 3:303-306.

[8] Sanjai Rayadurgam and Mats P.E. Heimdahl, "*Coverage Based Test-Case Generation using Model Checkers*", Proceedings Eighth Annual IEEE International Conference & Workshop On the Engineering of Computer-Based System s-ECBS 2001; 2001, p83-91, 9p.

[9] Katsuya Iwata, "*Black-box test case generation from TFM module interface specifications and usage statistics*", Iowa State University, 2012-01-01T08:00:00Z.

[10] Hyojoung Shin, Don-Jung Choi, Bo-Keong Kim, Taebok Yoon and Jee-Hyong Lee, "A Wear-leveling Scheme for NAND Flash Memory based on Update Patterns of Data", *Journal of the Korean Institute of Intelligent Systems*, 20(6), December 2010, 761-767

저 자 소 개

손명규(Myeong-Gyu Son)

2004년 : 울산대학교 재료공학과 학사
2005년~현재 : 삼성전자 근무
2014년~현재 : 성균관대학교 정보통신공학부 석사과정

관심분야 : SQA, 기계학습, 데이터 마이닝
E-mail : mg5son@skku.edu

이지형(Jee-Hyong Lee)

1993년 : 한국과학기술원 전산학과 학사
1995년 : 한국과학기술원 전산학과 석사
1999년 : 한국과학기술원 전산학과 박사
2002년~현재 : 성균관대학교 정보통신공학부 교수

관심분야 : 퍼지이론, 지능시스템, 기계학습
E-mail : john@skku.edu